Distributed System Abstractions for Multicores Systems

Frank Mueller
North Carolina State University
mueller@csc.ncsu.edu

ABSTRACT

Current trends in microprocessors are to steadily increase the number of cores. While multicores offer tremendous opportunities to meet processing demand, they come at the expense of limited scalability due to on-chip (interconnect) and off-chip (memory) resource contention. As the core count increases, current system and programming abstractions, such as pure task-level parallelism and single-image operating systems, become an obstacle rather than an aid in harnessing multicore power.

This work promotes a distributed system design, instead of the traditional shared memory view on a chip. It utilize mesh-based network-on-chip (NoC) communication. It provides a new abstraction layer of micro-kernels on few cores and pico-kernels on the vast majority of cores on a chip. This novel layer mitigates high-level node parallelism and low-level shared memory. It has the potential to be transparent to programmers but can also be explicitly addressed for bare-metal performance.

1. INTRODUCTION

The future of computing is rapidly changing as multicore processors are becoming ubiquitous. While multicores offer tremendous opportunities to meet processing demand, they come at the expense of limited scalability due to on-chip (interconnect) and off-chip (memory) resource contention.

Contemporary shared memory techniques have been shown to fall short in scaling, particularly at the system level where a single system image (SSI) remains the traditional abstraction. SSI was a good match for bus-based multiprocessors in the past. However, bus-based designs do not scale well (even beyond four processors) and have been replaced by mesh interconnects (e.g., Hypertransport, Quick Path Interconnect) and, for high core counts, tile-based architectures with 2D meshed network-on-chip (NoC) interconnects [2, 3, 8, 7, 1].

Mesh-based systems with MESI-style (Modified/Exclusive/Shared/Invalid) coherence protocols enhanced by coherence filters [6] may limit scalability in in the number of cores. For example, the multikernel (aka. Barrelfish) follows a distributed kernel paradigm that employs messages in an off-chip mesh interconnect of Hypertransport links [4]. It shows that messaging can outperform shared memory for configurations of just eight processors.

2. OPEN QUESTIONS

The hypothesis of this position paper is: *On the path to exascale, research is needed to develop novel system and program abstractions at the runtime and operating system layers that replace the shared-memory SSI legacy with an asynchronous message-based, decentralized and distributed system design, yet at small system size suitable for per-core deployment.* A pico kernel is envisioned that handles task dispatching and on-chip communication on each core. These per-core pico kernels would be controlled by micro kernels that govern the scheduling of tasks and coordinate higher-level operating system abstractions (e.g., off-chip device access, file systems). We seek to support a massive number of cores on a single chip, which paves the path to scalable runtime/operating systems for the future. We promote a set of novel system and program abstractions at the runtime and operating system layers that replaces the shared-memory SSI legacy with an asynchronous message-based, decentralized and distributed system design, yet at small system size suitable per-core deployment.

In the following, a high level description of the system design objectives and their relation to open questions in multicore research is provided.

Ensure predictive execution that maximizes the use of all available cores: An exascale core OS needs to combine low-cost scheduling with predictable process execution to facilitate load balancing. The following open problems need to be addressed: How does scheduling in a massive multicore environment differ from traditional operating system scheduling? How can concepts such as interrupts, priority, swapping be implemented while ensuring predictive execution? How can resource contention be reduced, both for NoC messages and off-chip traffic (memory, I/O)?

Guarantee scalability as the number of cores increases: Such an OS has t o ensure that performance increases linearly with the number of cores as more cores are added (scale out). The following open problems need to be addressed: What are the appropriate metrics or measures of effectiveness to evaluate performance? Are some measures biased towards specific applications (e.g., real-time image processing)? How can the shared memory limitations that plague some multicore systems be overcome? For example, does the potential of message passing among core elements scale to massive numbers of cores? Can application performance scale as core counts increase and NoCs become subject to contention? How can hardware routing be complemented by software re-routing to better utilize the NoC?

Provide an environment for power-awareness: The exascale OS needs to tackle the power consumption / leakage problems inherent to multicore systems. The following open problems need to be addressed: What is the interaction of scheduling with core activation? Is there any latency that needs to be accounted for? Are there tradeoffs between activating cores vs. attempting to maximize the use of cores already activated? How many cores should be utilized at a time to obtain an optimal energy-delay trade-off?

Develop runtime/operating systems abstractions for processors with massive numbers of cores: The envisioned exascale OS is designed as a "natural" fit with multicore systems, to be synergistic with, and complement them. This is as opposed to the current state of the art where operating systems are derived from legacies that were never initially designed from the ground up to accommodate high degrees of parallelism among tens or hundreds of processors or cores. The following open problems need to be addressed: What traditional operating system services are applicable to multicores (e.g. pipes, threads, mutexes, interrupts, signals, etc.)? How can they be tailored for multicore support, while maximizing the use of all the cores? How do language abstractions support such OS mechanisms for massive multi-cores?

Our vision is a system where featherweight layers of parallelism are scalably coordinated in a distributed manner to provide close to raw performance while ensuring predictability.

3. EXPERIMENTAL RESULTS

3.1 Microenchmarks

We have conducted experiments to assess the trade-offs between message passing and shared memory of a 64-core NoC (Tilera TilePro64) [3], which has multiple mesh networks (on chip). In a bandwidth micro-benchmark, we measured the transfer time in cycles for different data sizes. We compared message passing over the user dynamic network (UDN) with shared memory transfers over the coherence interconnect. Figure 1 indicates that shared memory incurs roughly twice the cost of message passing transfers (both without hashing). UDN messages follow a one-sided push model (sender initiated) while shared memory accesses are pull based (receiver initiated) and require at least two messages for a single transfer. The differences between shared memory and message passing become even more significant as the distance (hop count) between cores in the NoC increases and as NoC contention increases. *These results indicate that message passing has the potential to outperform shared memory transfers and that the former has superior scaling characteristics than the latter.*

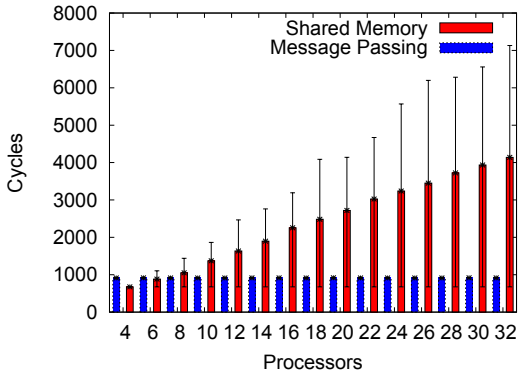

Figure 1: Increase in Shared Memory Contention/Jitter

The figure also indicates a significant increase in jitter (variability) for shared memory access latencies as the number cores and thus contention increases. These results are further supported by the increase in memory latencies caused by contention to shared memory of Figure 2. In a shared-memory system design, both the operating system and the application increasingly suffer from such latencies as the core count increases. *Due to these findings, we promote a system design that breaks with the shared memory abstraction.* While this is our primary philosophy, we would still allow shared memory abstractions for smaller ensembles of cores when performance and predictability allow to do so (for up to 16 cores).

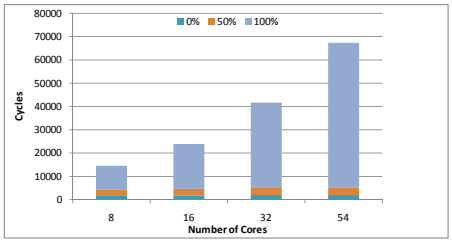

Figure 2: Shared Memory Latency for Contention Levels

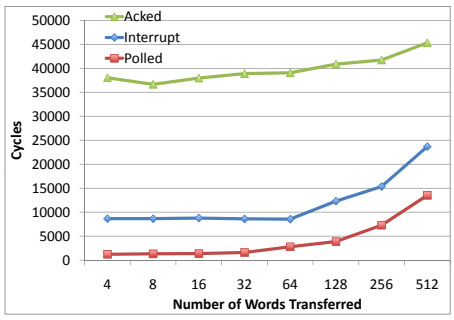

Figure 3: Message Latency for Different Protocols

3.2 NAS IS

We have implemented a parallel bucket sort algorithm (derived from the NAS IS benchmark) on the Tilera TilePro 64 over (a) shared memory and (b) message passing utilizing the vendor's interrupt-based approach. Figure 4 depicts the performance results (in billions on cycles) over an increasing number of cores (4-56). This is a weak scaling experiment [5] where the number of keys per core is fixed at a certain size. This ensures that the computational work per core remains the same as the number of cores cooperating in a parallel sort is increased. We measure the performance for per-core sizes of 2k, 4k 8k and 16k keys to ensure that all keys fit into the local L2 cache after a warm-up phase. The expected result of weak scaling under perfect scalability would be a flat line.

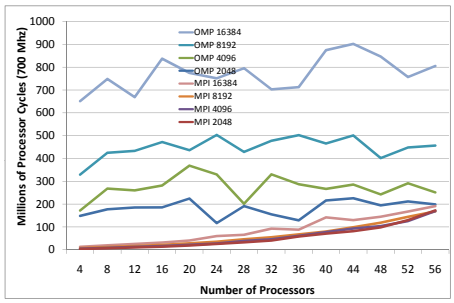

Figure 4: Performance of Bucket Sort for Different Problem Sizes

We make the following observations: (1) Shared memory inflicts overheads up to 50 times higher than message passing, especially for a small number of cores. This is due to the coherence proposal overhead and the latencies over the distributed L2, which causes frequent messages. (2) As the core count increases, overheads of message passing increase quadratically. This shows that the sorting algorithm is communication bound. The quadratic growth is due to increasing contention over the 2D mesh interconnect. (In a 3D mesh, the increase would be cubical.) (3) As the core count increases, overheads of shared memory vary erratically due to NoC contention. The variance for shared-memory results over all core sizes is in the same order as the difference between 4 and 56 cores for message passing, albeit without any erratic variance for the latter. By correlating these results with the vendor's acknowledgment-based message passing to our polling-based approach (Figure 3), we conjecture that communication performance can be improved up to 16 times for the parallelized bucket sort. Furthermore, by reducing contention over the NoC, our approach can result in superior scaling, i.e., a reduction from a quadratic overhead to a logarithmic one depending on the NoC overlay structure.

4. REFERENCES

[1] Single-chip cloud computer. blogs.intel.com/research/2009/12/sccloudcomp.php.

[2] Tera-scale research prototype: Connecting 80 simple sores on a single test chip. ftp://download.intel.com/research/platform/terascale/tera-scaleresearchprototypebackgrounder.pdf.

[3] Tilera processor family. http://www.tilera.com/products/processors.php.

[4] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. The multikernel: a new os architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, Symposium on Operating Systems Principles, pages 29–44, 2009.

[5] John L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, May 1988.

[6] Andreas Moshovos, Gokhan Memik, Alok Choudhary, and Babak Falsafi. Jetty: Filtering snoops for reduced energy consumption in smp servers. In *International Symposium on High Performance Computer Architecture*, pages 85–96, 2001.

[7] K. Sankaralingam, R. Nagarajan, P. Gratz, R. Desikan, D. Gulati, H. Hanson, C. Kim, H. Liu, N. Ranganathan, S. Sethumadhavan, S. Sharif, P. Shivakumar, W. Yoder, R. McDonald, S.W. Keckler, and D.C. Burger. The distributed microarchitecture of the trips prototype processor. In *International Symposium on Microarchitecture*, November 2006.

[8] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, and Anant Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27:15–31, 2007.